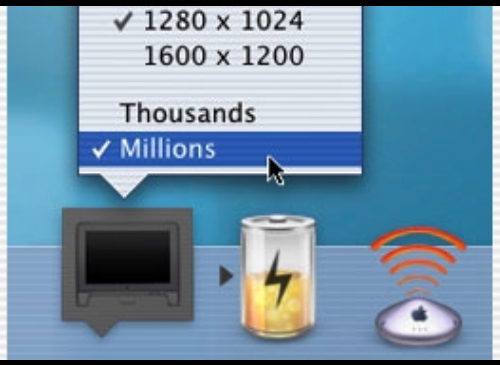
Session 116

Mac OS X
Core File Systems

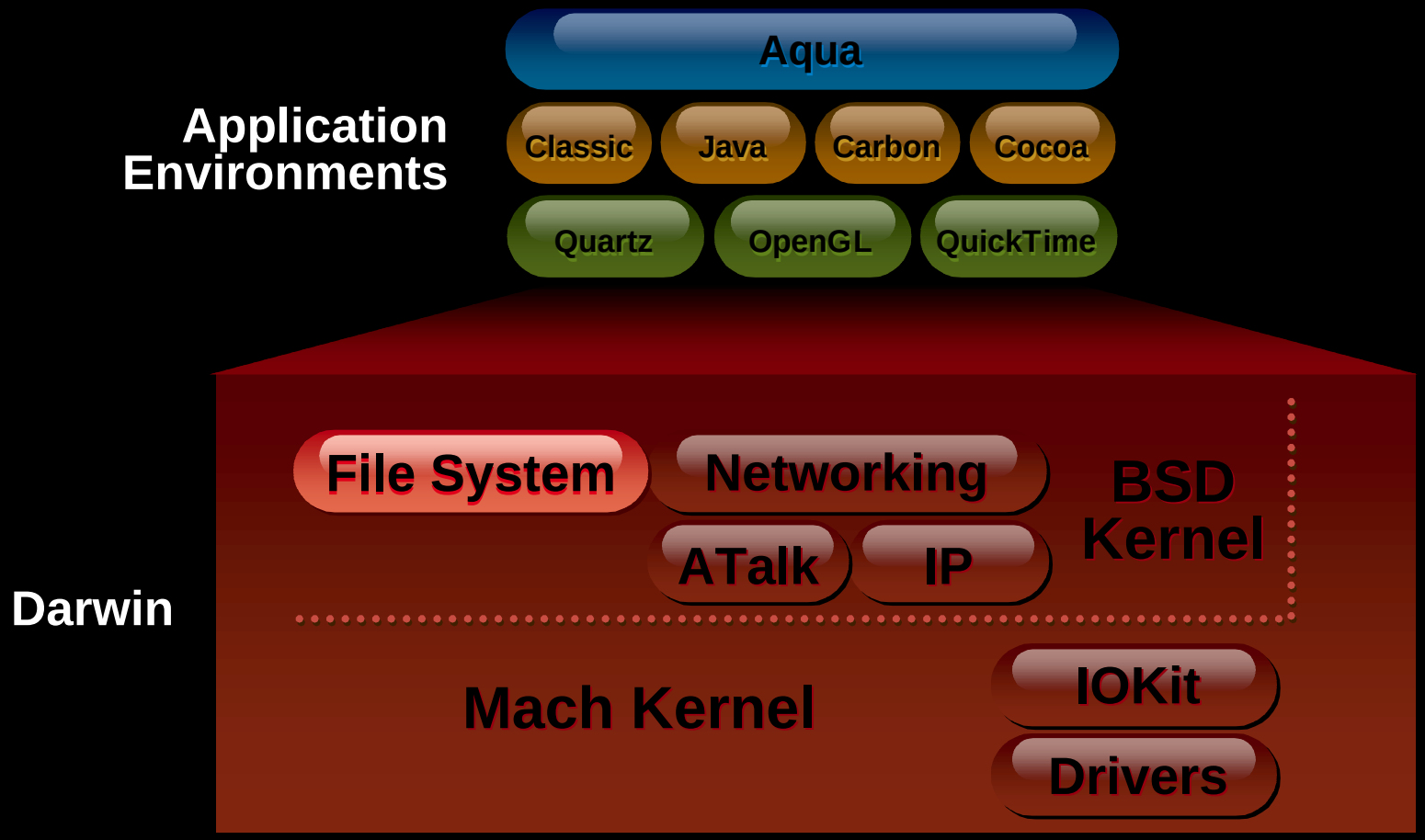


Clark Warner
Manager Core OS File Systems Engineering

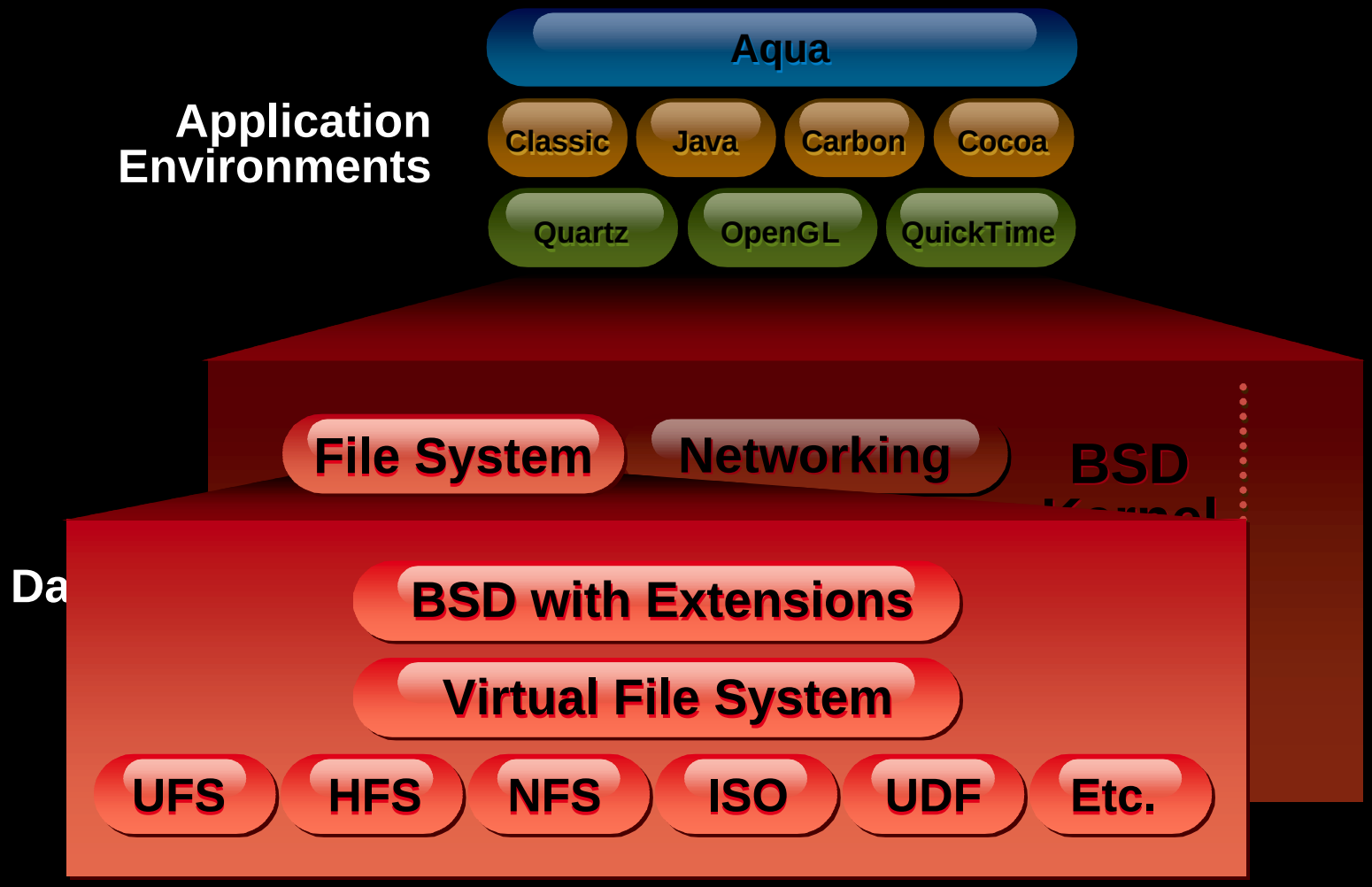
Welcome

- File system support in Mac OS X
  - What apps need to be ready for
    - File system differences
    - Key developer info
  - Making your app faster
    - Pointers on using the file system
  - Building your own file systems
    - Key issues

You Are Here


Application Environments
Aqua
Classic
Java
Carbon
Cocoa
Quartz
OpenGL
QuickTime
Darwin
File System
Networking
ATalk
IP
BSD Kernel
Mach Kernel
IOKit
Drivers

File System


Aqua
Application Environments
Classic
Java
Carbon
Cocoa
Quartz
OpenGL
QuickTime
File System
Networking
BSD
Da
BSD with Extensions
Virtual File System
UFS
HFS
NFS
ISO
UDF
Etc.

Outline

- Status Update
- Demo
- File System Interfaces
- File System Differences
- Security
- Performance Considerations
- Building File Systems

We Shipped!

Mac OS X File Systems

- Primary (Boot ,Root,Read/Write)
  - Mac OS Extended (HFS Plus), UFS, NFS
- Read/Write Only
  - HFS, AFP, MSDOS FS, WebDAV
- Read Only
  - ISO9660, UDF, CDDAFS

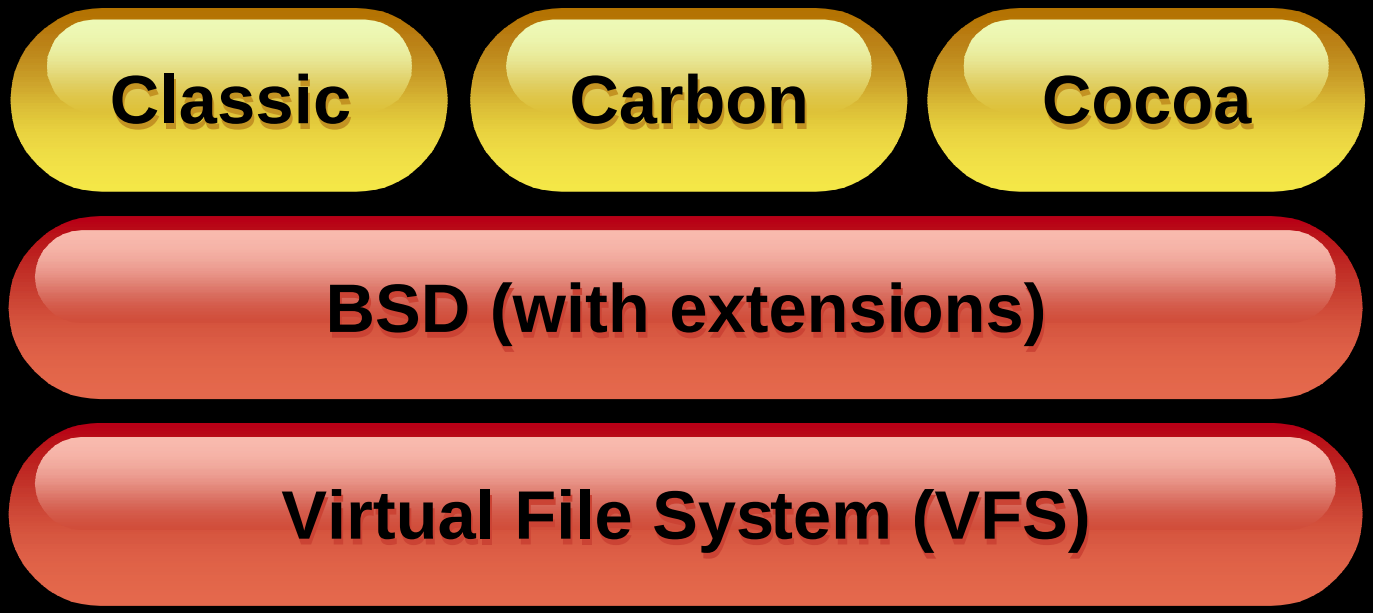
Demo

Clark Warner
Manager Core OS File Systems Engineering

Outline

- Status Update
- Demo
- File System Interfaces
- File System Differences
- Security
- Performance Considerations
- Building File Systems

File System Interfaces



File System Interfaces

- Carbon File Manager
  - Mac OS interface carried forward
  - Access by vRefNum/DirID/name
  - Provides HFS semantics on non-HFS file systems
- Cocoa
  - Object Oriented API
- BSD
  - Common access point for all file system services

New BSD System Call Interfaces

- **getattrlist()/setattrlist()**
  - Flexible way to get at all HFS/HFS + file information
- **searchfs()**
  - Implements full **CatSearch()** functionality
- **exchangedata()**
- **fcntl()**
  - New options for controlling space allocation

Outline

- Status Update
- Demo
- File System Interfaces
- File System Differences
- Security
- Performance Considerations
- Building File Systems

File System Differences

- Naming differences
  - Case and name length
- Access by persistent ID or path
- Delete of open files
- Unicode support
- Permissions

File System Differences

- Hard links
- Catalog data
- Multifork

WebDAV

- File system based on HTTP with extensions
  - Modification dates only
  - Inode numbers persistent during mount only
  - Silent failure when setting FS attributes
  - HTTP security model
  - Can be slow (link dependent)

File System Differences

	Privileges	Delete	ID's
HFS Plus	●	●	●
UFS	●	●	●
NFS	●	●	●
AFP	● / ●	●	●
UDF	●	●	●
WebDAV	●	●	●
ISO	●	●	●
MSDOS	●	●	●

File System Differences

	Case Sen	Unicode
HFS Plus	🔴	🟢
UFS	🟢	🔴
NFS	Depends	Depends
AFP	🔴	🔴
UDF	🟢	🟢
WebDAV	Depends	🔴
ISO	🔴	🔴
MSDOS	🔴	🔴

Managing Differences

- Be consistent in your use of case
- Always use decomposed unicode names
- Be prepared for access errors
- Do not rely on inode numbers across mounts

How Can You Tell?

- **pathconf()** system call
  - Case sensitivity
  - Maximum file name length

Outline

- Status Update
- Demo
- File System Interfaces
- File System Differences
- Security
- Performance Considerations
- Building File Systems

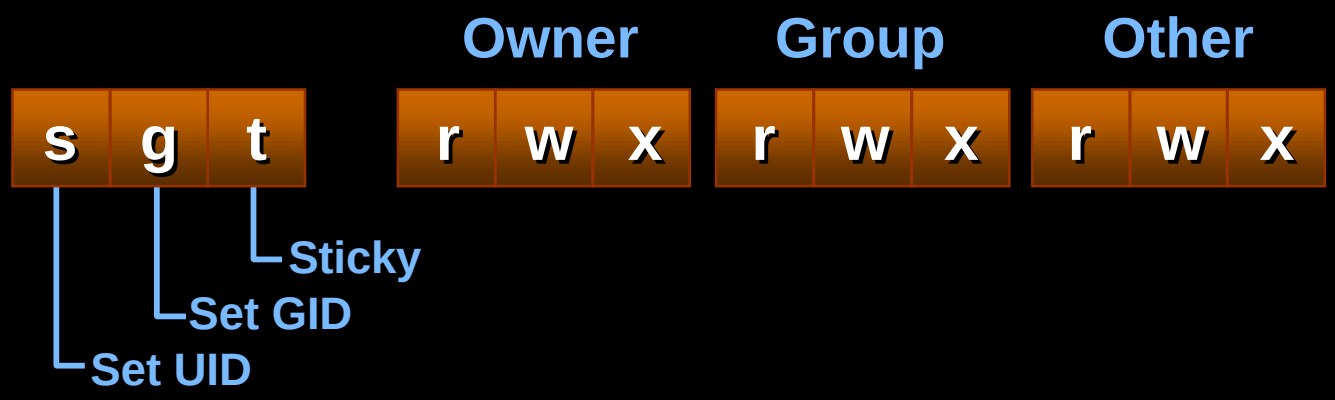
Session 116

Mac OS X Core File Systems

Pat Dirks
Core OS File Systems Technical Lead

Security in Mac OS X

- Mac OS X is a multi-user system
- Standard UNIX permissions on files and directories:



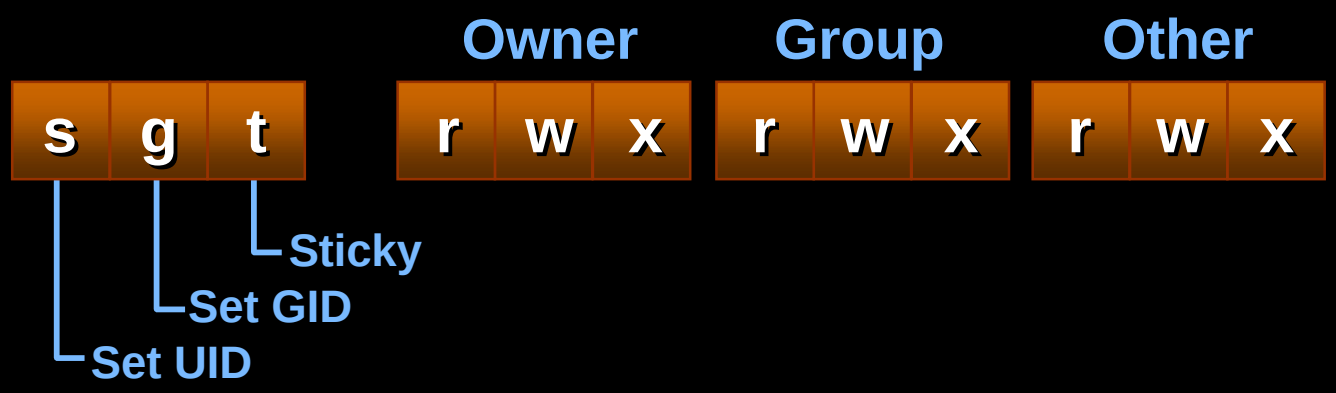
- Similar to AppleShare permissions model
- Special features on HFS/HFS+ volumes

Standard UNIX Permissions

- Similar to AppleShare
  - "Read/Write/Execute" instead of "See Files/See Folders/Make Changes"
- Applied to all folders and files
- Only single category of access applies

Standard UNIX Permissions

- Every file and folder has
  - Owner ID
  - Group ID
  - Permission bits (‘Read’/‘Write’/‘Execute’)
  - Special bits and flags



Standard UNIX Permissions

- Separate permissions for 3 categories of users:
  - Owner
  - Group
  - Others
- Most specific match determines category
  - Only single category's permissions apply

Permissions on Files

- For each category (owner/group/others):
  - ‘Read’
  - ‘Write’
  - ‘Execute’
- ‘Set UID’ runs program as owner
- ‘Set GID’ runs program as group

Permissions on Directories

- For each category (owner/group/others):
  - ‘Read’: permission to list directory contents
  - ‘Write’: permission to make changes in directory
  - ‘Execute’: permission to look up and access files by name in directory
- ‘Sticky bit’ limits ‘Write’ to owner of object

Flags: Immutability and Locking

- Single set of flags per file or folder
  - IMMUTABLE (replaces file locked bit)
  - APPEND
  - NODUMP
- Immutable files/folders cannot be moved
- Special system-only variants cannot be reset when system is up

New Features in Mac OS X HFS

- Special handling provided for permissions on removable HFS + media
- Local user has choice to use or ignore owner/group for local HFS + hard disks

Permissions on HFS+ Removable Media

- Handling governed by per-disk preference (every disk is identified by 64-bit unique ID)
- Default handling ignores owner/group on disk
- Local user can choose to use or ignore owner/group information on any local disk

“Ignore Permissions”

- Finder lets user choose to ignore all owner/group information of any local HFS/HFS + disk:
  - Owner becomes “unknown” (logged-in user)
  - Group becomes “unknown”
  - Permissions are used as-is with new owner/group
- Think of it as “Ignore Ownership”

Outline

- Status Update
- Demo
- File System Interfaces
- File System Differences
- Security
- Performance Considerations
- Building File Systems

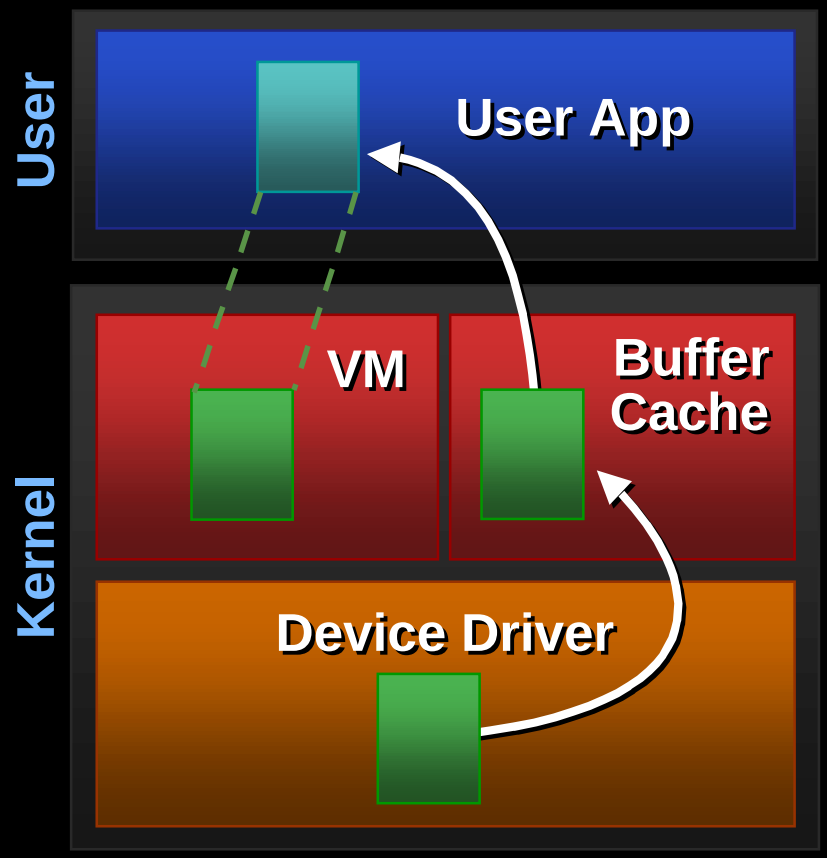
Optimizing I/O Performance

- General Considerations
- 3 ways to do I/O
  - Buffered File System I/O
  - Memory-mapped I/O
  - Unbuffered File System I/O
- Avoiding Zero fill on HFS/HFS + volumes
- The Cost of Caching
- Performance Analysis Tools

I/O Performance: General

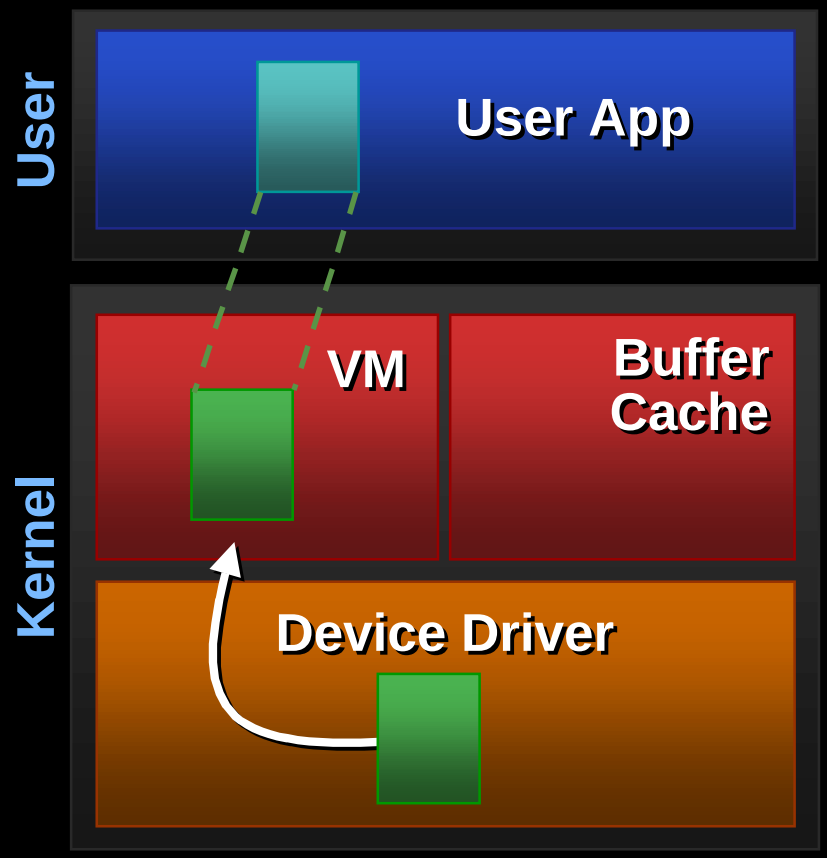
- Minimize the number of I/O operations
  - Fewer, larger transfers are always better than many smaller transfers
- The kernel transparently clusters I/O operations—Sequential access is always better than random access
- Avoid zero fill on write—don't skip ahead

Buffered File System I/O



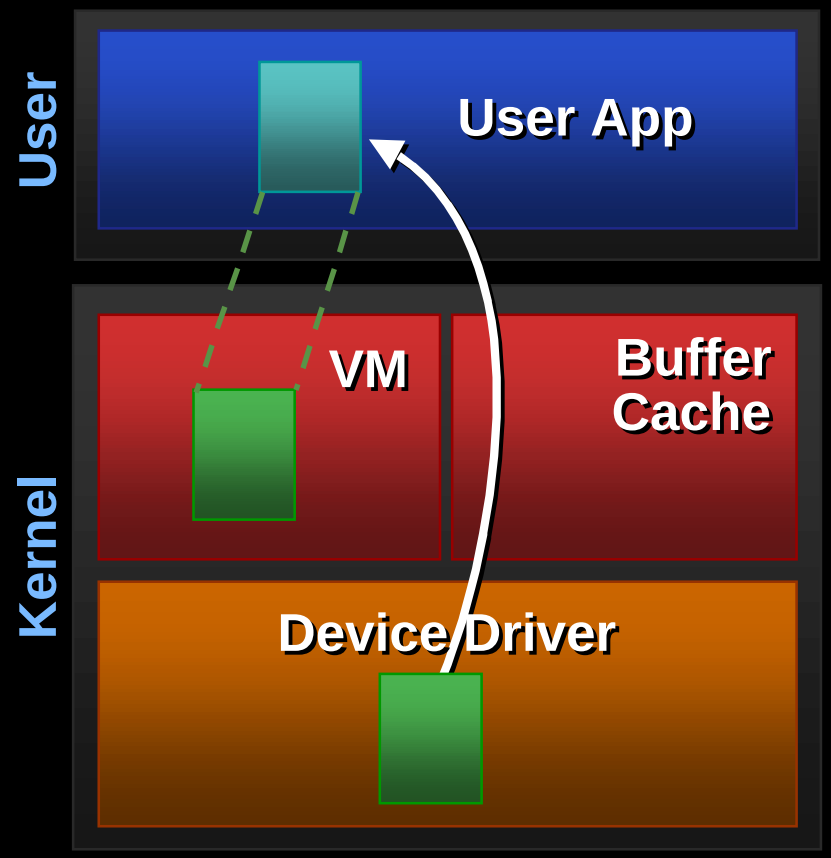
- Most flexible: arbitrary transfer size and alignment
- 2 copies of all data during transfer
- Destination page is dirtied in transfer
- Good for data that will be used again later

Memory-mapped I/O



- Requires BSD VM system call
- Every transfer at least 4K
- Only single copy of data
- All pages are left clean
- Good for input; can overwrite but cannot extend mapped file

Unbuffered I/O



- Full control over I/O transfer size
- Transfer must be 4K page-aligned
- Involves only 1 copy
- Target page is dirty
- Good for generating data not likely to used again soon

Avoiding Zero-fill

- Kernel prevents reading of unwritten disk blocks by zero-filling gaps
- Expanding file or writing starting beyond end-of-file triggers automatic zero-filling by system on HFS/HFS+ disks
- Sequential write is optimal

The Cost of Caching

- In-memory caches may force three I/O operations (page out/data read/page out later)
- Careful planning of cache data structures is vital: every data access could trigger a page-in from disk
- Caching information can become more expensive than re-reading it from disk!
- Don't bother unless cost to reconstruct data is actually high and hit rates in practice are > 90%

Recommendations

- Consider the nature of the data and pattern of access
- Analyze your application in action to determine bottlenecks
- Go back to the underlying assumptions behind original design
- Avoid making assumptions about local/remote nature of data and consequent speed of access!

Performance Analysis Tools

- Standard BSD UNIX tools:
  - ‘top’
  - ‘time’ command
- Mac OS X only tools:
  - ‘sample’
  - ‘fs_usage’

Demo

Clark Warner
Author of code bad enough to demo

Outline

- Status Update
- Demo
- File System Interfaces
- File System Differences
- Security
- Performance Considerations
- Building File Systems

Building Filesystems

- Not recommended
  - Potential for kernel instability
  - Contact us
- Requires deep internal knowledge
- Rev lock with kernel will be enforced

Recent Kernel Change

- Making Mac OS X preemptable
  - Mach is preemptable
  - BSD kernel historically is not

Funnels

- Mechanism to provide mutual exclusion
  - Acquired when thread enters system call
  - Dropped when thread returns to user mode
  - Held across kernel preemption
  - Dropped when thread yields in kernel

Funnel Types

- Network Funnel
  - Held when entering network system calls
    - E.g., Socket, bind, accept
- Kernel Funnel
  - Held for all other kernel calls including file system calls
- Either Funnel
  - mbufs, memory allocation and free, mach services

Kernel Need to Knows

- Unified Buffer Cache
- Split Funnel
- Enhancements to SMP support
- All the stuff we can’t cover in 1 hour

Roadmap

115 Open Source at Apple	Room A2 **Wed., 10:30am**
310 AFP 3.0 & AppleShare Client on Mac OS X	Room J2 **Thurs., 3:30pm**
121 Carbon Performance Tuning	Hall 2 **Thurs., 2:00pm**
705 Apple Performance Tools	Room A2 **Thurs., 5:00pm**
139 Leveraging BSD Services in Mac OS X	Civic **Fri., 2:00pm**

Roadmap

140 The Darwin Kernel	Civic **Fri., 3:30pm**
FF121 Feedback Forum: Darwin	Civic **Fri., 5:00pm**

Who to Contact

Jason Yeo
Core OS Technology Manager
Apple Worldwide Developer Relations
jason@apple.com

Q&A

Jason Yeo
Core OS Technology Manager
Apple Developer Relations
jason@apple.com

http://developer.apple.com/events/urls.html

WWDC

Worldwide Developers Conference 2001